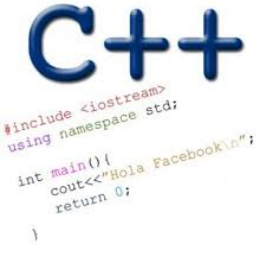
HEAPS

Problem Solving with Computers-II



Reminders

BST - movie dataset
Part 1. Answer questions about the data
Part 2. Running Time Analysis

- PA02 released, due Wed of Week 10 (06/02).
- Lab05 due this Wed (05/19)
- zyBook, Chapter 7 activities due this Friday (05/21)
- Quiz 4 next week Mon (05/24): zybook chapters 5, 6, 7 (Run Time Analysis, Stack, Queue, STL)

Heaps

- Clarification
  - *heap*, the data structure is not related to *heap*, the region of memory
- What are the operations supported?
- What are the running times?
- How do we implement it?

min-heap

Fast · insert

min

delete min

max-heap

insert

max

delete max

STL

push()

top()

pop()

empty()

Running Time

no. of keys in the heap

$O(\log N)$

$O(1)$

$O(\log N)$

Heaps

	Min-Heaps	Max-Heap
• Insert :	O(log N)	O(log N)
• Min:	O(1)	—
• Delete Min:	O(log N)	—
• Max	—	O(1)
• Delete Max	—	O(log N)

Any BST
O(N)
O(N)
O(N)
O(N)
O(N)

Balanced BST
O(log N)
O(log N)
O(log N)
O(log N)
O(log N)

Black box.

min-Heap

push(5)
push(20)
push(7)
top()//5
pop()

Applications:

- Efficient sort
- Finding the median of a sequence of numbers
- Compression codes

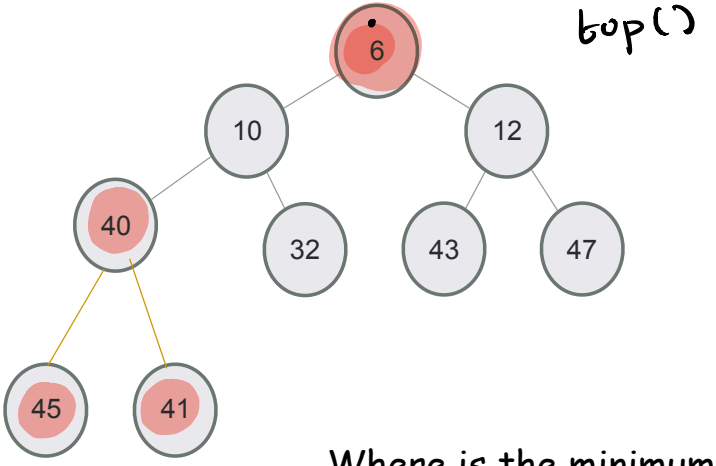
Choose heap if you are doing repeated insert/delete/(min OR max) operations

Heaps as binary trees

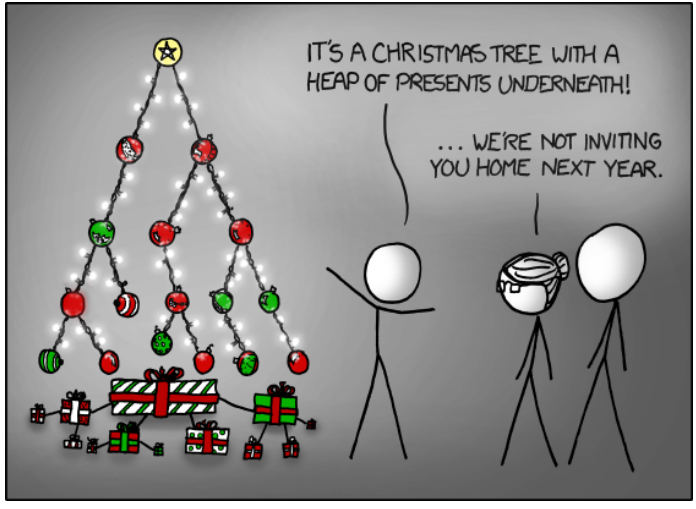
- **Rooted binary tree that is as complete as possible**
- **In a min-Heap, each node satisfies the following heap property:**

  key(x)<= key(children of x)

Min Heap with 9 nodes



top()

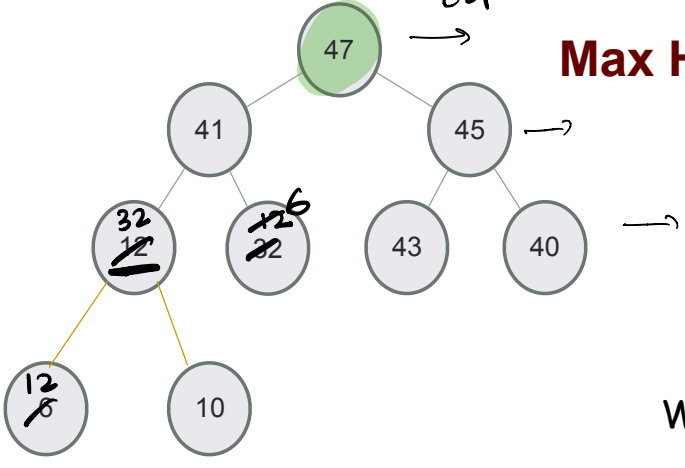


Where is the minimum element?

Heaps as binary trees

- **Rooted binary tree that is as complete as possible**
- **In a max-Heap, each node satisfies the following heap property:**

  key(x)>= key(children of x)

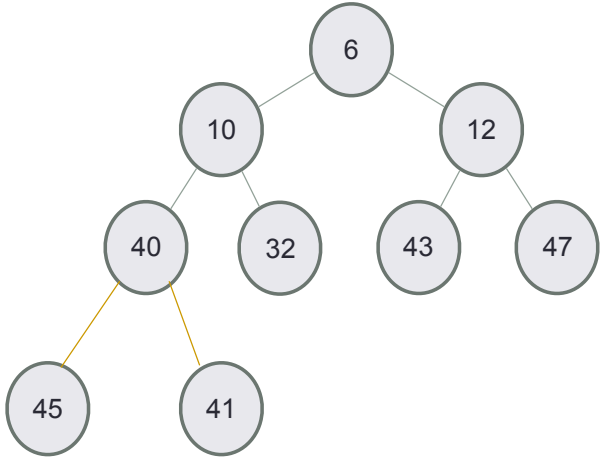


Max Heap with 9 nodes

Where is the maximum element?

Structure: Complete binary tree

A heap is a complete binary tree: Each level is as full as possible.
Nodes on the bottom level are placed as far left as possible



insert
push() O(log N)

Identifying heaps

Starting with the following min-Heap which of the following operations will result in something that is NOT a min Heap

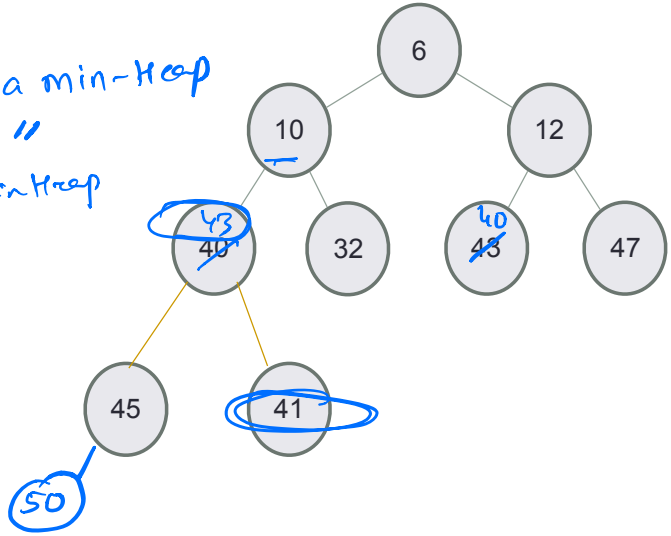
A. Swap the nodes 40 and 32

B. Swap the nodes 32 and 43

C. Swap the nodes 43 and 40

D. Insert 50 as the left child of 45

E. C&D

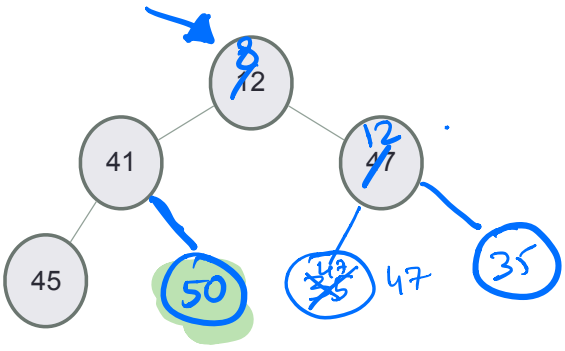


Insert 50 into a heap

insert(35) insert(8)

- Insert key(x) in the first open slot at the last level of tree (going from left to right)
- If the heap property is not violated - Done
- Else: while(key(parent(x))>key(x)) swap the key(x) with key(parent(x)) [Bubbling up]

min-Heap.

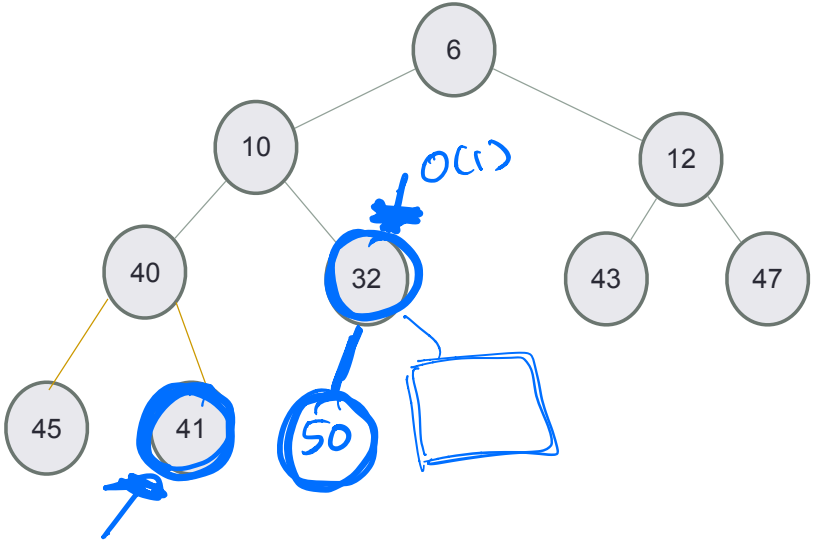


O(H)

O(logN)

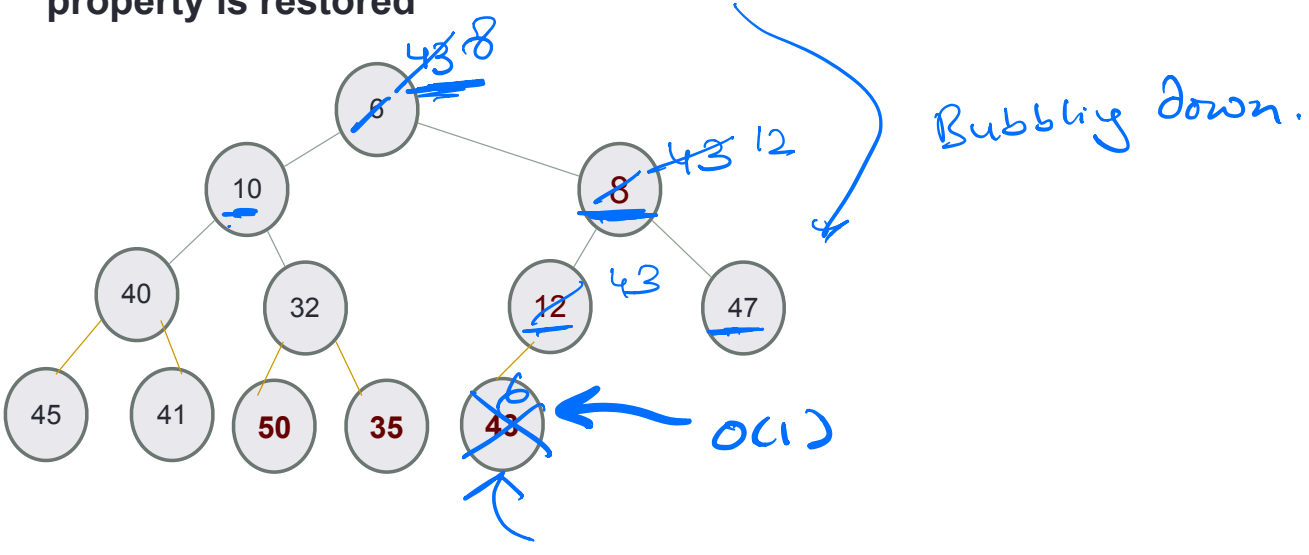
Insert 50 as the right child of 47 in O(1) (Magic?)

Insert 50, then 35, then 8



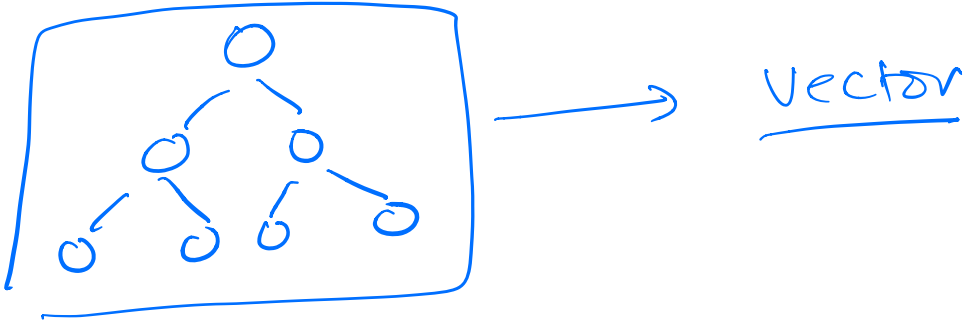
Delete min

- **Replace the root with the rightmost node at the last level**
- **“Bubble down”- swap node with one of the children until the heap property is restored**



Under the hood of heaps

- An efficient way of implementing heaps is using vectors
- Although we think of heaps as trees, the entire tree can be efficiently represented as a vector!!

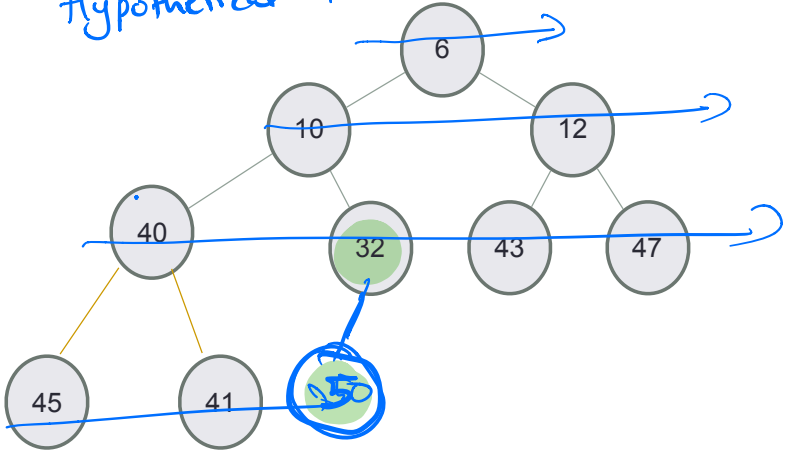


Implementing heaps using an array or vector

Value	6	10	12	40	32	43	47	45	41	50
Index	0	1	2	3	4	5	6	7	8	9

O(1)

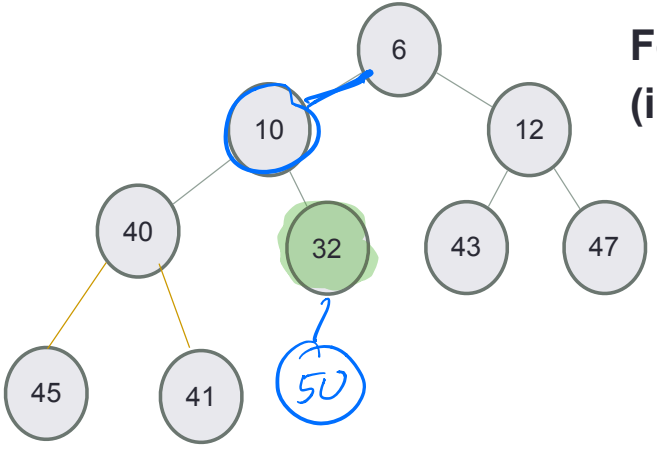
Hypothetical Tree min Heap



Using vector as the internal data structure of the heap has some advantages:

- **More space efficient than trees**
- **Easier to insert nodes into the heap**

Finding the "parent" of a "node" in the vector representation



For a key at index i, index of the parent is (i-1)/2

$\text{IndexP}(i) = \left(\frac{i-1}{2}\right)$

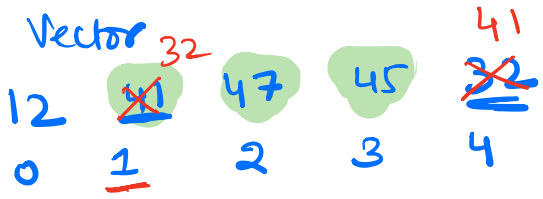
$\frac{9-1}{2} = 4$

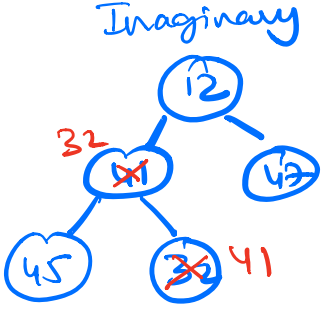
Value	6	10	12	40	32	43	47	45	41	50
Index	0	1	2	3	4	5	6	7	8	9
Index of Parent	—	0	0	1	1	2	2	3	3	4

Insert into a heap

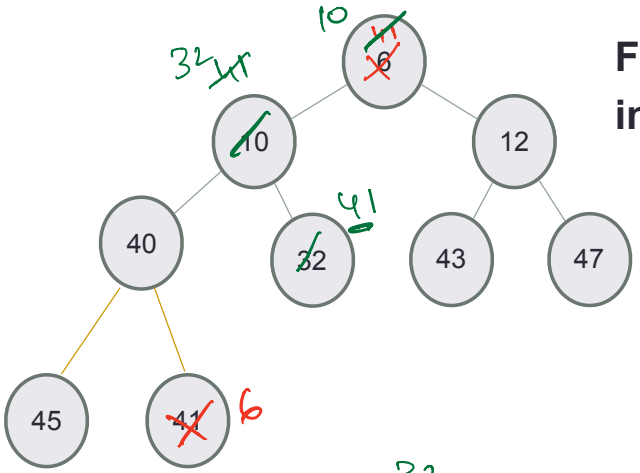
- Insert key(x) in the first open slot at the last level of tree (going from left to right)
- If the heap property is not violated - Done
- Else….

Insert the elements {12, 41, 47, 45, 32} in a min-Heap using the vector representation of the heap





~~Insert 50, then 35~~ Delete 6



For a node at index i, index of the parent is int(i-1/2)

$\left(\frac{i-1}{2}\right)$

Children (i)

i = 4

2i + 1 → 9 Left child

2i + 2 → 10 Right child

vector

v.size() − 1

Value	~~6~~	~~10~~	12	40	~~32~~	43	47	45	~~41~~
Index	0	1	2	3	~~4~~	5	6	7	~~8~~
LeftChild	1	3	5	7	~~9~~				
RightChild	2	4	6	8	10				

index out of bound means no right child.

Insert 8 into a heap

Value	6	10	12	40	32	43	47	45	41	50	35
Index	0	1	2	3	4	5	6	7	8	9	10

After inserting 8, which node is the parent of 8 ?

A. Node 6

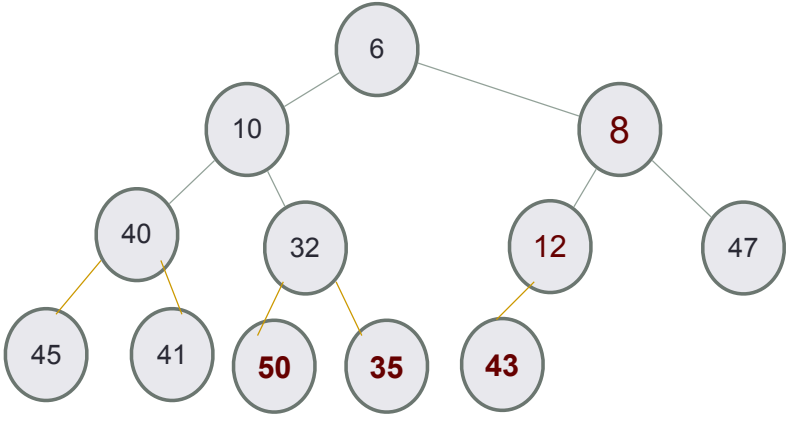
B. Node 12

C. None 43

D. None - Node 8 will be the root

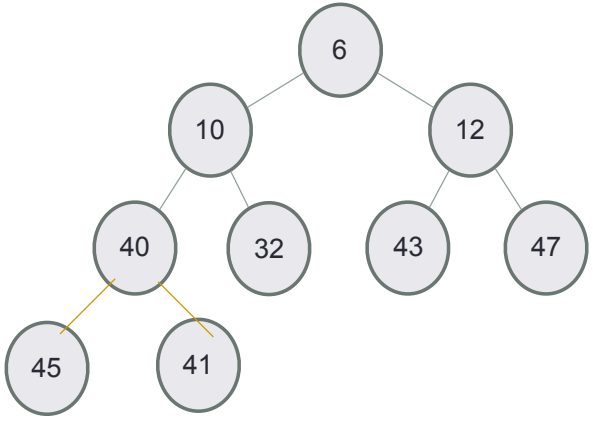
Delete min

- **Replace the root with the rightmost node at the last level**
- **“Bubble down”- swap node with one of the children until the heap property is restored**



Traversing down the tree

Value	6	10	12	40	32	43	47	45	41	
Index	0	1	2	3	4	5	6	7	8	



For a node at index i, what is the index of the left and right children?

A. (2*i, 2*i+1)

B. (2*i+1, 2*i+2)

C. (log(i), log(i)+1)

D. None of the above

Next lecture

- **Under the hood of heaps**
- **More on STL implementation of heaps (priority queues)**